



Алгоритмы и структуры данных

Лекция 15. Алгоритмы на графах

Антон Штанюк (к.т.н, доцент)

26 мая 2022 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра "Компьютерные технологии в проектировании и производстве"

Основные определения

Программные реализации

Обход графов

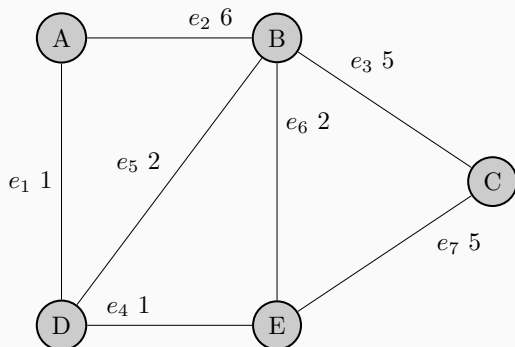
Список литературы

Основные определения

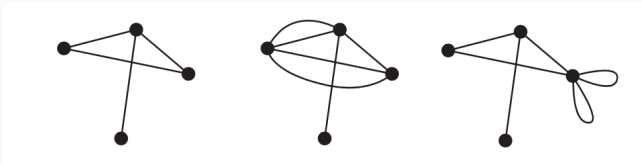
Граф - это абстрактное представление множества объектов и связей между ними. Графом называют пару (V, E) где V - это множество вершин, а E - множество пар, каждая из которых представляет собой связь (эти пары называют рёбрами).

Граф может быть *ориентированным* или *неориентированным*.

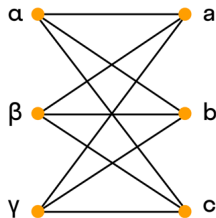
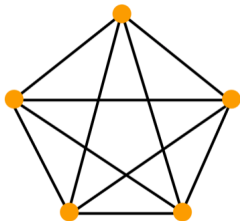
Путь в графе это конечная последовательность вершин, в которой каждые две вершины идущие подряд соединены ребром. Путь может быть ориентированным или неориентированным в зависимости от графа.



Простой, мультиграф, псевдограф



Полный и двудольный графы



Два ребра называются *смежными*, если у них есть общая вершина.

Два ребра называются *кратными*, если они соединяют одну и ту же пару вершин.

Ребро называется *петлей*, если его концы совпадают.

Степенью вершины называют количество ребер, для которых она является концевой (при этом петли считают дважды).

Вершина называется *изолированной*, если она не является концом ни для одного ребра.

Вершина называется *висячей*, если из неё выходит ровно одно ребро.

Граф без кратных ребер и петель называется *обыкновенным*.

Циклом называют путь, в котором первая и последняя вершины совпадают.

Путь или цикл называют *простым*, если ребра в нем не повторяются.

Если в графе любые две вершины соединены путем, то такой граф называется *связным*.

Два графа называются *изоморфными*, если у них поровну вершин. При этом вершины каждого графа можно занумеровать числами так, чтобы вершины первого графа были соединены ребром тогда и только тогда, когда соединены ребром соответствующие занумерованные теми же числами вершины второго графа.

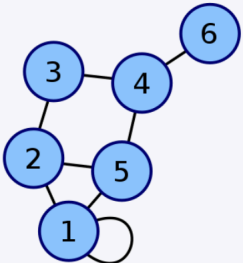
Граф **H**, множество вершин **V'** которого является подмножеством вершин **V** данного графа **G** и множество рёбер которого является подмножеством рёбер графа **G** соединяющими вершины из **V'** называется *подграфом* графа **G**.

Программные реализации

1. Матрица смежности
2. Список смежности
3. Матрица инцидентности
4. Список ребер

Матрица смежности

По строкам и столбцам номера вершин, а значения в матрице говорят, связаны ли соответствующие вершины ребрами

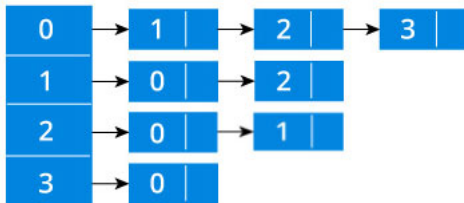
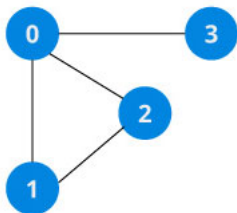
Граф	Матрица смежности
	$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$

Матрица инцидентности

Строки соответствуют вершинам от 1 до 6, а столбцы — рёбрам e1–e7.

Граф	Матрица инцидентности
	$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Создается массив список соединенных вершин



Матрица смежности удобна, когда граф плотный (ребер больше чем вершин). Для разреженных графов в матрице будет много нулей.

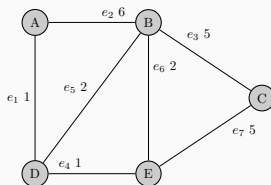
Преимуществом матрицы смежности является быстрота проверки соединения i -й вершины с j -й вершиной.

Список смежности подходит для разреженных графов (число ребер меньше или равно числу вершин). Памяти требуется существенно меньше, чем при использовании матрицы смежности. Недостатком является время поиска конкретного ребра.

Класс Graph на основе матрицы смежности

```
class Graph {  
private:  
    std::vector<std::vector<int>> matrix; // матрица смежности  
    int vnumber; // число вершин  
public:  
    Graph(int vn):vnumber(vn) {  
        matrix.resize(vnumber, std::vector<int>(vnumber));  
    }  
    void display() { // вывод матрицы на экран  
        int i, j;  
        for(i = 0; i < matrix.size(); i++) {  
            for(j = 0; j < matrix[0].size(); j++) {  
                std::cout << matrix[i][j] << " ";  
            }  
            std::cout << std::endl;  
        }  
    }  
    void add_edge(int u, int v) { //функция добавления ребра в матрицу  
        matrix[u][v] = 1;  
        matrix[v][u] = 1;  
    }  
};
```

Сконструируем граф:



Пример задания графа

```
int main(int argc, char* argv[]) {  
    enum VER {A,B,C,D,E};  
    Graph gr(5);  
    gr.add_edge(A,B);  
    gr.add_edge(A,D);  
    gr.add_edge(B,D);  
    gr.add_edge(B,E);  
    gr.add_edge(B,C);  
    gr.add_edge(C,E);  
    gr.display();  
    return 0;  
}
```

```
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 0
0 1 1 0 0
```

Класс Graph на основе списков смежности

```
class Graph {  
    private:  
        std::map<int, std::list<int>> adj; // список смежности  
        int vnumber; // число вершин  
    public:  
        Graph(int vn):vnumber(vn) {}  
        void display() {  
            for(auto& [key, value] : adj) { // C++17!  
                std::cout << key << "└─┐";  
                for(auto& i: value) {  
                    std::cout << i << "┐";  
                }  
                std::cout << std::endl;  
            }  
        }  
        void add_edge(int u, int v) { // функция добавления ребра в матрицу  
            adj[u].push_back(v);  
            adj[v].push_back(u);  
        }  
};
```

Пример задания графа

0 - 1 3

1 - 0 3 4 2

2 - 1 4

3 - 0 1

4 - 1 2

Обход графов

Основными методами обхода являются:

1. Обход в глубину (DFS)
2. Обход в ширину (BFS)

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина. С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

Для поиска в глубину и в ширину создаем отдельные функции, которые должны быть объявлены дружественными классу **Graph**.

Функция DFS будет рекурсивной. Для того чтобы хранить посещённые вершины, создается статический массив, доступ к которому обеспечивается только из копий функции DFS.

Функция BFS - не рекурсивная.

```
void DFS(Graph& gr, int v, bool first) {
    static std::map<int, bool> visited;
    if(first) {        // первый запуск
        visited.clear();
        first = false;
    }
    visited[v] = true;
    std::cout << v << "␣";
    for (auto i = gr.adj[v].begin(); i != gr.adj[v].end(); ++i)
        if (!visited[*i])
            DFS(gr, *i, false);
}
```






Поиск в ширину (BFS)






```
void BFS(Graph& gr, int s) {
    std::vector<bool> visited;
    visited.resize(gr.vnumber, false);
    // создаем очередь BFS
    std::list<int> queue;
    // помечаем текущий узел как посещённый
    visited[s] = true;
    queue.push_back(s);

    while(!queue.empty()) {
        // извлекаем из очереди и печатаем
        s = queue.front();
        std::cout << s << "␣";
        queue.pop_front();

        for (auto adjacent: gr.adj[s]) {
            if (!visited[adjacent]) {
                visited[adjacent] = true;
                queue.push_back(adjacent);
            }
        }
    }
}
```

Список литературы

-  Кормен Т., Лейзерсон Ч., Ривест Р.
Алгоритмы: построение и анализ
МЦНМО, Москва, 2000
-  Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы:
построение и анализ.
2-е изд. — М.: «Вильямс», 2006
-  Википедия
Алгоритм
<http://ru.wikipedia.org/wiki/Алгоритм>
-  Википедия
Список алгоритмов
http://ru.wikipedia.org/wiki/Список_алгоритмов
-  Традиция
Задача коммивояжёра
<http://traditio.ru/wiki/Задача>

-  Википедия
NP-полная задача
<http://ru.wikipedia.org/wiki/NP-полная>
-  Серджвик Р.
Фундаментальные алгоритмы на C++. Части 1-4
Diasoft, 2001
-  Седжвик Р.
Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
СПб.: ДиаСофтЮП, 2003
-  Седжвик Р.
Фундаментальные алгоритмы на C. Алгоритмы на графах
СПб.: ДиаСофтЮП, 2003
-  Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.
Издательский дом «Вильямс», 2000



Кнут Д.

Искусство программирования, том 1. Основные алгоритмы
3-е изд. — М.: «Вильямс», 2006



Кнут Д.

Искусство программирования, том 2. Получисленные методы
3-е изд. — М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 3. Сортировка и поиск
2-е изд. — М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений
М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации

М.: «Вильямс», 2007